Hans-Jürgen Lange <hjl@simulated-universe.de>

March 10, 2020

**Abstract**

This is a description of the ideas behind the development of the modeler. As the world alreadys has a lot of graphical modelers, and for povray some editors as well, what would be the benefit of a new modeler.

# 1 Intention

So first I must admit, I am not a graphics designer. But I have a lot of fun with let some objects move around in a video. povray comes with some good support to handle animations, but the more complex the movement grows the more bulky povray with its scripting language will be.

And even with only a view movements in a scene from some point on you need more and more computational power and it all is very uneasy to handle.

The original kpovmodeler has some good ideas in that it split up the whole scene into its components in a tree and allows the designer to change almost everything that you would do in the povray source.

But it has no support for animations and so your still stuck at that.

# 2 Animations!

For to handle animations it needs more than to define a starting and ending frame number. As an animation typically shows up in a larger scene with objects each as complex as in the rendering of single frame you have to handle far more resources.

A single animation is fun. But it is often part of a collection of scenes that together will make up a complete video.

So it make sense to allow multiple scenes that share the resources that are used.

Working on multiple scenes may involve several designers. Maybe one does some fancy textures while others may create a crater on a moon.

So it would be good to allow multiple designers to work on the animation at the same time.

The third point that animations may require is computational power.

For a simple preview of a single object the local OpenGL or povray rendering may be sufficient. But to create a complete scene, or all at once, will need some sort of render farm, that needs to be managed.

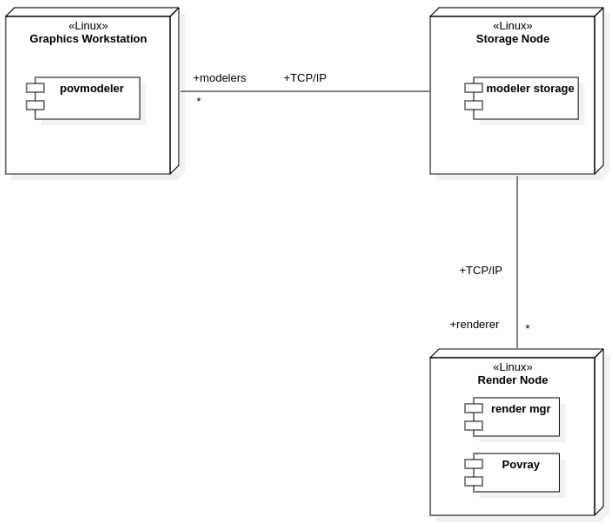


Figure 1: How it may be

To allow multiple users to work on the model it is stored in some sort of database. This database is not even responsible to handle the data but to handle the communication to the render managers on the different render nodes as well. So the modeller software only does its communication with the storage node.

The storage node has all information available to handle the different render nodes.

As all data will be stored in the storage node the render nodes only need access to the data stored there. At best all data is stored in the database and gets streamed from there to the render nodes.

# 3 Modeller

The modeller will have a GUI very much like the kpovmodeller. The first implementation will use some code from the storage node to use it as a standalone solution.

And anyways we creating here all anew the modeller should be able to read the old kpovmodeler files. So for a first try there is no need to use the storage node software.

# 4 Storage Node

I did create already a software that could handle the model. It is able to manage multiple modelers and renderers as well. But it is somehow larger with its

components and I must have a look on the source what parts to make available in the next steps.

### 4.1 Model Classes

The model classes are generated with my very own code generation tool. The model store normaly runs as part of a larger application environment. But even here it is perfectly suited to handle the model items.

Because the code gets generated there is much more in there as we may need. On the other hand the code need no checking as it is not produced by copy&paste. So I do not expect bugs from copying code.

As one of the basic classes I defined a CObjectBase class.

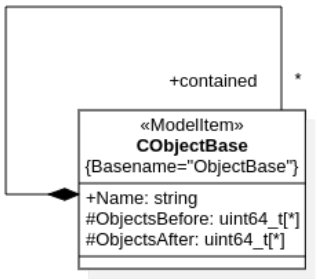


Figure 2: The base object

The linkage between an object and the contained ones is generated into the code. It works even if the objects are stored in a database.

As the model items are fully abstract the compiler does not know what kind of model item is contained. It may be anything from with in the model store.

Another benefit of the generated code is the ability to define templates of the classes that can be used to create new instances. As the two lists for restricting the placement of a model item into the tree depend on the model item itself we can define these lists in the templates. As long as we use the templates for the instantiation of new model items we will always have these list copyed.

This works best with the database as storage as we may update the templates and with the next loading of the database content we have new placement restrictions.

## 5 Render Manager

The render manager is a command line tool that handles the communication with the storage node. This way the storage node knows what machines are connected for what purpose.